Machine Learning Engineer Nanodegree

Reinforcement Learning

Project 4: Smartcab

A smartcab is a self-driving car from the not-so-distant future that ferries people from one arbitrary location to another. In this project, you will use reinforcement learning to train a smartcab how to drive.

Implement a basic driving agent

Implement the basic driving agent, which processes the following inputs at each time step:

- Next waypoint location, relative to its current location and heading,
- Intersection state (traffic light and presence of cars), and,
- Current deadline value (time steps remaining),
- Produces some random move/action (`None`, `forward`, `left`, `right`).

Don't try to implement the correct strategy! That's exactly what your agent is supposed to learn.

Run this agent within the simulation environment with enforce_deadline set to False (see run function in agent.py), and observe how it performs. In this mode, the agent is given unlimited time to reach the destination. The current state, action taken by your agent and reward/penalty earned are shown in the simulator.

Question 1

In your report, mention what you see in the agent's behavior. Does it eventually make it to the target location?

Answer: The agent moved randomly in each direction if I added random actions for the agent. It would reach the target location in the last sometimes, but the deadline was exceeded almost every time.

Identify and update state

Identify a set of states that you think are appropriate for modeling the driving agent. The main source of state variables are current inputs, but not all of them may be worth representing. Also, you can choose to explicitly define states, or use some combination (vector) of inputs as an implicit state. At each time step, process the inputs and update the current state. Run it again (and as often as you need) to observe how the reported state changes through the run.

Question 2

Justify why you picked these set of states, and how they model the agent and its environment.

Answer:

The states I used are:

- The traffic light `light`: ['red', 'green']
- The traffic at the coming straight `oncoming`: [None, 'forward', 'left', 'right']
- The traffic at the coming left `left`: [None, 'forward', 'left', 'right']
- The traffic at the coming right `right`: [None, 'forward', 'left', 'right']
- Next waypoint location `next_waypoint`: [None, 'forward', 'left', 'right']

The goal of this project is to train the cab can reach the target and obey the traffic rules.

The `light`, `oncoming`, `left`, and `right` states are related to traffic rules, the agent will get reward or penalty according to whether it obeyed the traffic rules. The traffic rules are:

- `light`: ['red'], the agent can turn right if `oncoming` state is not ['left'] or `left` state is not ['forward'].
- `light`: ['green'], the agent can turn left only if `oncoming` state is not ['forward'] .

`next_waypoint` state is the best action calculated by route planner, the agent should obey the action from `nex_waypoint` as well to reach the target point as soon as possible.

`deadline` is another state related with the reward. The agent can not get the final reward 10 if it can not reach the target within deadline time. However, this state will affact the performance when I added it into the set of sates.

Implement Q-Learning

Implement the Q-Learning algorithm by initializing and updating a table/mapping of Q-values at each time step. Now, instead of randomly selecting an action, pick the best action available from the current state based on Q-values, and return that. Each action generates a corresponding numeric reward or penalty (which may be zero). Your agent should take this into account when updating Q-values. Run it again, and observe the behavior.

Question 3

What changes do you notice in the agent's behavior?

Answer: The agent moved randomly firstly, and then it obeyed the traffic rule, and followed the actions provided by `next_waypoint` state in the last.

However, the agent can not reach the target within deadline time and won't obey the derection provided by route planner sometimes befor some parameters tuned(as shown in next question).

Enhance the driving agent

Apply the reinforcement learning techniques you have learnt, and tweak the parameters (e.g. learning rate, discount factor, action selection method, etc.), to improve the performance of your agent. Your goal is to get it to a point so that within 100 trials, the agent is able to learn a feasible policy - i.e. reach the destination within the allotted time, with net reward remaining positive.

Question 4

Report what changes you made to your basic implementation of Q-Learning to achieve the final version of the agent. How well does it perform?

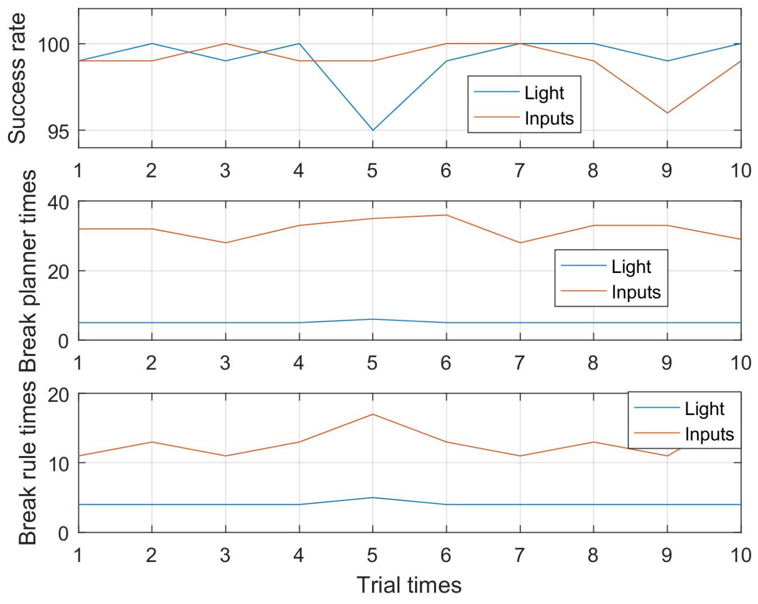
Answer:

- Action selection method is picking the available action with max Q-value as the best action from the current state based on Q-values.
- Learning rate determines to what extent the newly acquired information will override the old information.
    - Learning rate is 0, it will make the agent not learn anything, the Q-values do not update, the success rate is about 0.
    - Learning rate is 1, there is a very high success rate, which is 98/100 ~ 100/100 from 10 trails.
- Discount factor determines the importance of future rewards.
    - It closes to 0 leads to "myopic" evaluation, while close to 1 leads to "far-sighted" evaluation.

In this project, the learning rate should be chosen as 1 because it fully deterministic environment. The discount factor should be selected a lower value to reduce the influence of high reward at the final target, because the agent can get the the target with `net_waypoint` action and the destination changed every time.

When the states only contain `light` and `next_waypoint` states: The success rate is high and the traffic rule breaking times and planner action breaking times are very low. But when the learning rate less than 1, the success rate is not stable.

When the states contain all the sates of `inputs` and `next_way_point`, the success rate is stable and very high. However, the raffic rule breaking times and planner action breaking times are quite high in the same time.



I wonder why the few states contained, the performance is better. So I increased the trail times from 100 to 1000, and found that:

- when the states contained `light` and `next_waypoint`, the success rate is still high(999/1000), the traffic rule breaking time is increased rarely (the probability is about 0.3%), the reason is the agent did not know the overall traffic rules, it will only concer the red and green light, but not concern whether there is traffic at the intersection. when I

added extra 10 cars on the road, the probability of traffic rules breaking increased to 0.3%~2%.
- when the states contained all `inputs` states and `next_waypoint`, the performance did not improved. The success rate is about 998/1000, The traffic rule and planner action breaking times were high and increased intermittently as well. The reason could be Q-values too widely dispersed due to the high dimensions.

I selected the `light` and `next_waypoint` as states, learning rate is 1. discount factor is 0 in the last.

Question 5

Does your agent get close to finding an optimal policy, i.e. reach the destination in the minimum possible time, and not incur any penalties?

Answer: My agent can get close to finding an optimal policy with a high success rate and lower faults number. But there is still few probability to make faults on traffic rules and planner actions breaking .

It doed not reach the destination in the minimum possible time, becasue the reward of reaching the target is same if the agent can reach the target before deadline. If we want to get the minimum possible time to reach the destination, it is better to add the relation between reward/penalty and time.